Chapter 31

Compression and Encryption Services for x900-48FE and AT-9900 Switches

Introduction

This chapter describes the data compression and data security services available on the switch, how the services are provided, the switch network functions that use these services, and how to monitor the services.

Data Compression

Data compression for switches is driven by the high cost of *wide area network* (WAN) access and user demands for increased bandwidth. The cost of WAN access is a significant part of the cost of providing a data network and the use of data compression on networks can result in significant savings. Compression increases the effective throughput of data across a network link by reducing the size of packets. This allows more packets to be transmitted over the link in the same time interval, or the same number of packets to be transmitted over a slower (and cheaper) link in the same time interval.

Data compression identifies redundancy in the data and produces an encoded form that is smaller, yet contains all the information required to recreate the original data. This is called *lossless data compression*, as opposed to voice and video compression which, due to their analog nature, normally use *lossy data compression* algorithms. Most modern high performance data compression techniques use variations of the Lempel-Ziv algorithm. This algorithm compresses data by maintaining data histories at the compression and decompression ends of the link. These histories contain the most recent data transmitted on a data link. The data to be compressed is compared against the history to find any common sequences. When a match is found, a reference to the position of the matching sequence in the history is sent instead of the data sequence itself. Compression is achieved because the reference is smaller than the sequence it represents. The algorithm is adaptive, adjusting automatically to produce the best compression ratio for the content of the data being compressed. A checksum is typically added to the data before compression to allow the validity of the data to be checked on decompression.

It is impossible to compress all possible data streams. A stream of totally random data has no redundancy and therefore cannot be compressed. Similarly, data that has already been compressed is unlikely to be further compressed. Some compression algorithms, such as the STAC LZS algorithm, cause incompressible data to expand during the compression process.

Network data compression can be organised into the following categories:

- **Link compression**
- **Payload compression**
- **Header compression**

Link compression

Link compression has traditionally been provided by an external device connected between a switch port and the WAN access device. The main disadvantages of external compression devices are that they require a separate connection to each switch port requiring compression and to each WAN access device, and they cannot be managed from within the switch's management structure.

Link compression operates by compressing the whole data stream, including the network layer packet headers used for routing. This means that the packet header is no longer accessible by intermediate devices that do not support the

particular compression algorithm. Even if an intermediate device does support the particular compression algorithm, packets must be decompressed and compressed at each device so that the packet headers can be read. This places an additional load on the device and results in high latency. Consequently, external link compression is normally only used in point-to-point configurations where the local and remote switches are directly connected, without any intermediate switches.

Integrating the compression function into the switch enables a single compression resource to support the compression of multiple links over any switch interface, replacing multiple external compression devices. Integration also allows the switch to support protocols, such as PPP multilink, which can spread data from one compression channel across multiple physical links. The compression process can be configured and monitored using the switch's own management interface, instead of a separate management system used only for the external compression device.

Payload compression

Payload compression is used to compress packet data at the network layer, without changing the packet header. Since the routing information remains unchanged the packet can be carried across a routed network, such as the Internet, without requiring the intermediate switches to support the compression algorithm or have any knowledge about how to access the compressed data.

Payload compression is usually not as efficient as link compression due to the fact that each packet must be compressed with no reference to any other packet—as packets may be lost or re-ordered while traversing the network. This means that the compression history must be cleared before compressing a packet, losing any advantage gained from compressing the previous packets. Only large packets or packets containing highly compressible data benefit greatly from payload compression.

The main benefit of payload compression over link-layer compression is in combination with payload encryption. Compressing data after it has been encrypted is a pointless exercise, as encrypted data is not compressible. When payload encryption is being used, payload compression can be performed before encryption, giving useful compression in some cases. Such functionality is difficult for an external device to achieve, since the device needs to understand the network layer protocol to determine which part of the packet to compress or decompress.

Header compression

Van Jacobson's header compression algorithm (defined in RFC 1144) can be used in TCP/IP networks to compress the standard 40-byte TCP/IP header of TCP packets down to 5 bytes (sometimes even down to 3 bytes). This produces a significant performance improvement when the majority of traffic consists of small packets. Because the switch processor must perform the compression calculations, this method is normally recommended for lower speed (less than 64Kbps) links. Van Jacobson's header compression applies only to TCP packets carried over Point-to-Point Protocol (PPP) links. ENCO is not required for header compression.

See Chapter 13, Internet Protocol (IP) for more information about configuring Van Jacobson's header compression.

Data Encryption

Data encryption for switches is driven by the need for organisations to keep sensitive data private and secure. Encrypting network data before it is passed to the wide area network (WAN) ensures that the data cannot be read or modified as it traverses the WAN. Since all wide area traffic passes through the switch, the switch is the ideal place to locate the complex hardware required to provide secure data encryption. Locating the encryption function in the network switch and integrating the complex encryption key management procedures into the switch's management system minimises the cost of supporting an encrypted network.

Data encryption operates by applying an encryption algorithm and key to the original data (the plaintext) to convert it into an encrypted form (the ciphertext). The ciphertext produced by encryption is a function of the algorithm used and the key. Since it is easy to discover what type of algorithm is being used the security of an encryption system relies on the secrecy of its key information. When the ciphertext is received by the remote device the decryption algorithm and key are used to recover the original plaintext. Often a checksum is also added to the data before encryption to allow the validity of the data to be checked on decryption.

The main classes of encryption algorithms are:

- **Symmetrical Encryption**
- **Asymmetrical (Public Key) Encryption**

Symmetrical Encryption

Symmetrical encryption refers to algorithms where a single key is used for both the encryption and decryption processes. Anyone who has access to the key used to encrypt the plaintext can decrypt the ciphertext. Because the encryption key must be kept secret to protect the data, these algorithms are also called private, or secret key algorithms. The key can be any value of the appropriate length.

DES encryption

The most common symmetrical encryption system is the *Data Encryption Standard* (DES) algorithm (FIPS PUB 46). The DES algorithm has withstood the test of time and proved itself to be a highly secure encryption algorithm. DES encryption does not require a require licence.

To fully conform to the DES standard, actual data encryption operations must be carried out in hardware. Therefore, software implementations are said to be *DES-compatible*, not DES-compliant. The DES algorithm in the switch has been optimised to produce very high speed hardware implementations, making it ideal for networks where high throughput and low latency are essential. The DES algorithm has a key length of 56 bits and operates on 64-bit blocks of data.

The following table describes DES modes.

DES Mode	Description
Electronic Code Book (ECB)	The fundamental DES function. Plaintext is divided into 64-bit blocks that are encrypted with the DES algorithm and key. For a given input block of plaintext, ECB always produces the same block of ciphertext.
Cipher Block Chaining (CBC)	The most popular form of DES encryption and the mode the switch uses. CBC also operates on 64-bit blocks of data, but includes a feedback step that chains consecutive blocks so that repetitive plaintext data (such as ASCII blanks) does not yield identical ciphertext. CBC also introduces a dependency between data blocks that protects against fraudulent data insertion and replay attacks. The feedback for the first block of data is provided by a 64-bit Initialisation Vector (IV).
Cipher FeedBack (CFB)	An additive stream cipher method that uses DES to generate a pseudo-random binary stream that combines with the plaintext to produce the ciphertext. The ciphertext is then fed back to form a portion of the next DES input block.
Output FeedBack (OFB)	Combines the first Initialisation Vector (IV) with the plaintext to form ciphertext. The ciphertext is then used as the next IV.

3DES encryption

The Triple DES (3DES) encryption algorithm is a simple variant on the DES CBC algorithm. The DES function is replaced by three rounds of that function, an encryption followed by a decryption followed by an encryption. This can be done by using either two DES keys (112-bit key) or three DES keys (168-bit key). Triple DES encryption requires a feature licence. For details contact your authorised distributor or reseller.

The two-key algorithm encrypts the data with the first key, decrypts it with the second key and then encrypts the data again with the first key. The three-key algorithm uses a different key for each step. The three-key algorithm is the most secure algorithm due to the longer key length.

Triple DES encryption can be performed in several modes. The following table describes the most common ones.

3DES Mode	Description
Inner CBC	Encrypts the entire packet in CBC mode three times, and requires three different initialisation vectors (IV's).
Outer CBC	Triple encrypts each 8-byte block of a packet in CBC mode three times and requires one IV.

Asymmetrical (Public Key) Encryption

Asymmetrical encryption algorithms use two keys—one for encryption and one for decryption. The encryption key is called the *public key* because it cannot be used to decrypt a message and therefore does not have to be kept secret. Only the decryption, or *private key*, needs to be kept secret. Another name for this type of algorithm is *public key encryption*.

The public and private key pair cannot be randomly assigned but must be generated together. In a typical scenario, a decryption station generates a key pair and then distributes the public key to encrypting stations. This

distribution need not be kept secret, but must be protected against the substitution of the public key by a malicious third party. Another use for asymmetrical encryption is as a digital signature. The signature station publishes its public key, and then signs its messages by encrypting them with its private key. To verify the source of a message the receiver decrypts the messages with the published public key. If the message that results is valid then the signing station is authenticated as the source of the message.

The most common asymmetrical encryption algorithm is RSA. RSA uses mathematical operations that are relatively easy to calculate in one direction but which have no known reverse solution. The security of RSA relies on the difficulty of factoring the modulus of the RSA key. Because typical key lengths of 512 bits or greater are used in public key encryption systems, decrypting RSA encrypted messages is almost impossible with current technology.

Asymmetrical encryption algorithms require enormous computational resources, making them very slow when compared to symmetrical algorithms. For this reason they are normally only used on small blocks of data, for example, exchanging symmetrical algorithm keys, and not for entire data streams.

Network Encryption

Network data encryption can be put into the following categories:

- **Link encryption**
- **Payload encryption**

Link encryption

Link encryption has traditionally been provided by an external device connected between a switch port and the WAN access device. The main disadvantages of external encryption devices are that they require a separate connection to each switch port requiring encryption and to each WAN access device, they cannot be managed from within the switch's management structure, and they do not normally support dial-up interfaces such as ISDN.

Link encryption operates by encrypting the whole data stream, including the network layer packet headers used for routing. This means that the packet header is no longer accessible by intermediate switches that do not support the particular encryption algorithm. Even if an intermediate switch does support the particular encryption algorithm, packets must be decrypted and re-encrypted at each switch so that the header can be read. This places an additional load on the switch and results in high latency. Consequently, external link encryption is normally only used in point-to- point configurations where the local and remote switches are directly connected without any intermediate switches.

Integrating the encryption function into the switch lets a single encryption resource support the encryption of multiple links over any switch interface, replacing multiple external encryption devices. Integration also allows the switch to support protocols such as PPP multilink, which can spread data from one encryption channel across multiple physical links. The encryption process can be configured and monitored using the switch's own management interface, instead of a separate management system used only for the external encryption device.

Payload encryption

Payload encryption is used to encrypt packet data at the network layer, without changing the packet header. Since the routing information remains unchanged the packet can be carried across a routed network, such as an internet, without requiring the intermediate switches to support the encryption

algorithm or have any knowledge about how to access the secure data. This allows non-secure internets to be used to form a highly secure network for the transport of sensitive information. As well as preventing unauthorised viewing and modification of the data, payload encryption can be used to prevent unauthorised access into the network. Only a switch that knows the secret encryption keys can access the network. All other access attempts decrypt incorrectly and are discarded. Such functionality is very difficult for an external device to achieve since the device would need to understand the network layer protocol before it could tell which part of the packet to encrypt or decrypt.

Data Authentication

Data authentication for switches is driven by the need for organisations to verify that sensitive data has not been altered. Authenticating network data before it is passed to the wide area network (WAN) ensures that the data cannot be altered as it traverses the WAN. Since all wide area traffic passes through the switch, the switch is the ideal place to locate the complex processing required to provide secure data authentication. Locating the authentication function in the network switch and integrating the complex authentication key management procedures into the switch's management system minimises the cost of supporting an authenticated network.

Data authentication operates by calculating a Message Authentication Code (MAC), commonly referred to as a *hash*, of the original data and appending it to the message. The MAC produced is a function of the algorithm used and the key. Since it is easy to discover what type of algorithm is being used the security of an authentication system relies on the secrecy of its key information. When the message is received by the remote switch another MAC is calculated and checked against the MAC appended to the message. If the two MACs are identical the message is authentic.

Typically a MAC is calculated using a keyed one-way hash algorithm. A keyed one-way hash function operates on an arbitrary-length message and a key and returns a fixed length hash. The properties that make the hash function one-way are:

- it is easy to calculate the hash from the message and the key
- it is very hard to compute the message and the key from the hash
- it is very hard to find another message and key that give the same hash

The two most commonly used one-way hash algorithms are MD5 (Message Digest 5, defined in RFC 1321) and SHA-1 (Secure Hash Algorithm, defined in FIPS-180-1). MD5 returns a 128-bit hash and SHA-1 returns a 160-bit hash. MD5 is faster in software than SHA-1, but SHA-1 is generally regarded to be slightly more secure.

HMAC is a mechanism for calculating a keyed Message Authentication Code that can use any one-way hash function. It allows keys to be handled the same way for all hash functions and it allows different sized hashes to be returned.

Another method of calculating a MAC is to use a symmetric block cypher such as DES in CBC mode. This is done by encrypting the message and using the last encrypted block as the MAC and appending this to the original message (plain-text). Using CBC mode ensures that the whole message affects the resulting MAC. See "DES encryption" on page 31-4 for more information about DES in CBC mode.

Key Exchange Algorithms

Key exchange algorithms are used by switches to securely generate and exchange encryption and authentication keys with other switches. Without key exchange algorithms, encryption and authentication session keys must be manually changed by the system administrator. If it is not possible to gain physical access to all switches in the secure network, it is virtually impossible to do this securely. Key exchange algorithms enable switches to re-generate session keys automatically and on a frequent basis.

The most important property of any key exchange algorithm is that only the negotiating parties are able to decode or generate the shared secret. Because of this requirement, public key cryptography plays an important role in key exchange algorithms. Public key cryptography encrypts a message that can be decrypted by only one party. A switch can generate a session key, encrypt the key using public key cryptography, transmit the key over an insecure channel, and be certain that the key can only be decrypted by the intended recipient. Symmetrical encryption algorithms can also be used for key exchange but commonly require an initial shared secret to be manually entered into all switches in the secure network.

ENCO on the Switch

ENCO provides services to user applications via channel pairs. A user application requests a service, specifying any configuration needed for the service, and is attached to an ENCO channel pair if the service and free channels are available. A channel pair consists of an encoding channel and a decoding channel. An encoding channel is used for compression, encryption, Diffie-Hellman key exchange, or authentication. A decoding channel is used for decompression, decryption, or authentication.

ENCO provides the following services:

- **Compression**
- **Encryption**
- **Authentication**
- **Key Creation and Storage**
- **Key Exchange**

Services may require a feature licence before they can be used. For specific information, contact your authorised distributor or reseller.

The number of channels available depends on the amount of RAM on the switch. Switches with up to 8MBytes of RAM can have up to 512 encryption and compression channels. Switches with 16MBytes can have up to 1024 channels, and switches with 32MBytes up to 2048 channels. To see the amount of RAM on a switch, use the **show system** command on page 4-54 of Chapter 4, Configuring and Monitoring the System.

To display the identification number of the lowest and highest channels available, use the **show enco** command on page 31-24. This command also displays general information about ENCO and the services that are available.

A user application that requests the retention of process histories between packets for an encryption or compression service (see "User Applications" on page 31-13) may also request that the history of one of its channels be reset. Whenever a decoding channel gets out of step with its associated encoding channel, the encoding channel's history must be reset.

Compression

ENCO provides the following compression algorithms:

- STAC LZS
- Predictor

Predictor compression is the default compression algorithm for PPP link compression. Predictor compression requires a lot of memory for its compression history and is not recommended for switches with less than 4MBytes of RAM.

You must configure the number of ENCO channels for software compression with the **set enco sw** command on page 31-23. This command must be run from a boot configuration script since software compression algorithms require contiguous memory, and the most efficient way to acquire it is just after the switch reboots. The maximum number of software compression channels is limited due to the large amount of memory that software compression algorithms require; STAC LZS requires 13KBytes per channel and Predictor requires 128KBytes per channel. No compression channels are configured by default.

STAC LZS compression is provided in software on the AT-8900 and AT-9900 switches; however, some ATI devices provide STAC LZS in hardware and software. Because the hardware and software processes are interoperable, you can configure STAC LZS on a link between a device that provides STAC LZS in software and one that provides it in hardware.

Encryption

ENCO provides DES and RSA encryption. The following variants of DES encryption are supported:

- 56-bit single DES
- 112-bit 2-key Outer CBC Mode Triple DES
- 168-bit Inner CBC Mode Triple DES
- 168-bit Outer CBC Mode Triple DES

The Triple DES variants require a special feature licence. Single DES encryption is available for the Secure Shell (SSH) and Secure Sockets Layer (SSL) protocols.

The switch requires Secure Shell (SSH) in order to use RSA encryption. For information about SSH, see Chapter 33, Secure Shell.

Authentication

ENCO provides the following authentication algorithms:

- HMAC MD5-96
- HMAC SHA-1-96

Key Creation and Storage

ENCO creates and stores keys the switch requires for encryption and authentication services. The following table describes keys that the switch uses.

This key...	Is used for...
General	HMAC authentication
DES	DES encryption
3DES2KEY	112-bit 2-key Outer CBC Mode Triple DES encryption
3DESINNER	168-bit Inner CBC Mode Triple DES encryption
RSA	RSA encryption

Keys are stored in flash memory. Each key has a unique identification number, and each one has a set of attributes, including the user application associated with the key. Each user application specifies whether these key attributes must be set. For example, SSH uses the IP address attribute of RSA keys to find the public key of the remote peer.

Creating keys

To create, import, or export encryption keys, use the **create enco key** command on page 31-14. This command requires a user with security officer privilege when the switch is in security mode. To display keys, use the **show enco key** command on page 31-45.

The generation of RSA keys is a time-consuming and processor-intensive operation. Two very large random prime numbers must be created and then combined to form the RSA key. The fastest method for generating large prime numbers is to create a large random number and then test to see if it is prime. If is not, then the number is modified and tested again. Because this is a very random procedure, it can take anywhere between 3 seconds and 30 minutes, depending on the size of the key and the CPU. To ensure the normal operation of the switch is not effected, RSA key generation is performed as a background task.

RSA public keys can be imported from and exported to text files. Keys generated by an external device, such as a PC, can be loaded onto the switch in text file format (`.key` files) and ENCO RSA public keys are generated from these text files. When an RSA key generated by the switch is exported to a text file, only the public portion of the key is written to the file. RSA keys cannot be entered via the command line.

File formats

Keys can be stored in the switch and displayed in several formats.

General keys can be entered in hexadecimal format or as a phrase that is easy to remember. The pass phrase must contain only printable characters and if spaces are included, the phrase must be in double quotes. General keys are displayed in hexadecimal format and also in string format if the key data contains all printable characters.

DES, 3DES2KEY, and 3DESINNER keys can be entered in a proprietary short/checked ASCII format or in hexadecimal format. The short/checked ASCII format represents each 5 bits of the key by an ASCII character. The legal characters are lowercase letters (a–z) and digits (2–9). The digits 0 and 1 are not used, to prevent confusion with the letters O and I. A checksum field is added to ensure the key has been entered correctly. DES, 3DES2KEY and 3DESINNER keys are displayed in both hexadecimal and the short/checked ASCII format.

The switch recognises the following text file formats:

- SSH (Figure 31-1)
- NIQ (Figure 31-2)
- HEX (Figure 31-3)—default

The first number in the key file is the length of the RSA public key in bits, the second number is the exponent field of the key and the last number is the modulus field of the key. In SSH format files the length, exponent, and modulus fields are all in one long line. In Figure 31-1 the "\" character shows where the line has been wrapped. In NIQ and HEX format files the length, exponent, and modulus fields are on separate lines.

Figure 31-1: RSA public key file in SSH format

```
512 65537 582714540409421726755748030187077328862507329405933811534665149936372 69\
25543081397311308147828977987913742520391622516348731783649991255114050692755958 77
```

Figure 31-2: RSA public key file in NIQ format

```
-NiQ Switch RSA Public Key
512
65537
582714540409421726755748030187077328862 5
073294059338115346651499363720692554308 1
397311308147828977987913742520391622516 3
487317836499912551140506927559587 7
```

Figure 31-3: RSA public key file in HEX format

```
512
0x010001
0x6f427e5112e1389e2af1c4df09545fa88f90b093aabbdebb5778ef5ed1d39fe9
248602ef11e399216b52adae2f5fd1ae8b7ca5c19b3c27a3ec5179966cb58465
```

Key Exchange

The switch uses the Diffie-Hellman key exchange algorithm, which is one of the more commonly used key exchange algorithms. It is not an encryption algorithm because messages cannot be encrypted with it. Instead, it provides a way for two parties to generate the same shared secret with the knowledge that no other party can generate that same value. It uses public key cryptography and is often considered the first public key algorithm. Its security is based on the difficulty of solving the "discrete logarithm problem", which can be compared to the difficulty of factoring very large integers.

A Diffie-Hellman algorithm requires more processing overhead than RSA based key exchange schemes, but does not need the initial exchange of public keys. Instead, it uses published and well tested public key values. The security of the Diffie-Hellman algorithm depends on these values. Public key values less than 768 bits long are not considered to be secure.

A Diffie-Hellman exchange starts with both parties generating a large random number. These values are kept secret, while the result of a public key operation on the random number is transmitted to the other party. A second public key

operation, this time using the random number and the exchanged value, results in the shared secret. As long as no other party knows either of the random values, the secret is safe. The exchange can be summarised in the following phases:

1. A local random number is generated, combined with a Diffie-Hellman public key value, and transmitted to the other party in the key exchange.
2. A shared secret is generated from the exchanged and local random number values.

Each phase is handled as smaller bits of processing because of the processor-intensive public key calculations. As a result the key exchange takes longer but the general operation of the switch continues during the process.

The Diffie-Hellman key exchange has a high priority by default. If the speed of the key exchange is not critical, the priority can be set to a lower value to leave more CPU time available for routing processes. To change the Diffie-Hellman key exchange priority, use the **set enco dhpriority** command on page 31-21.

ENCO supports published public key values "MODP Group 1" and "MODP Group 2" as defined in RFC 2412, *The OAKLEY Key Determination Protocol*. These public key values are 768 and 1024 bits long, respectively.

Access Control

When encryption is configured and enabled, the switch must be placed in *security mode* by using the **enable system security_mode** command on page 29-43 of Chapter 29, User Authentication.

Important Keys created on a switch that is not in security mode are destroyed when the switch is restarted. Enable security mode before creating encryption keys.

When the switch is in security mode, only users with security officer privilege can execute commands that could impact the security of the switch and its keys.

A user must login from a local port, a Secure Shell session, or a Telnet session from a remote security officer address (if the Remote Security Officer function is enabled) to gain security officer privilege. See Chapter 33, Secure Shell for more information about Secure Shell.

See Chapter 29, User Authentication for more information about creating users with security officer privilege, configuring remote security officers, and logging in to a user account with security officer privilege.

When an encrypting switch is removed from service, its sensitive encryption keys should be cleared before it is transported in an unprotected manner. All encryption keys on a switch can be cleared by disabling security mode by using the **disable system security_mode** command on page 29-39 of Chapter 29, User Authentication

User Applications

IP payload encryption and VPNs

Both AT-8900 and AT-9900 switches combine payload encryption technology and security associations to create secure *virtual private networks* (VPNs) across the Internet. They support IP compression using LZS compression as defined in RFC 2393 and RFC 2395. This method is a proprietary Security Association implementation that is supported for backward compatibility with Software Version 7.7.

Secure Shell

Secure Shell (SSH) requires the DES and RSA encryption algorithms. ENCO makes these algorithms available to the SSH.

For information about configuring Secure Shell, see Chapter 33, Secure Shell.

PPP

The PPP protocol can use ENCO services to provide link compression and/or link encryption.

The switch implements the Compression Control Protocol (CCP) as defined by RFC 1962 to provide compression on PPP. CCP provides a method for negotiating the compression algorithm to use and algorithm-specific parameters such as the check mode. It also provides a mechanism for synchronising the compression histories at each end of the link if they become unsynchronised. The use of STAC LZS and Predictor compression with PPP is defined in RFCs 1962 and 1978, respectively.

The switch implements the Encryption Control Protocol (ECP) as defined by RFC 1968 to provide encryption on PPP links. ECP provides a method for negotiating the encryption algorithm to use and algorithm-specific parameters. It also provides a mechanism for synchronising the encryption and decryption processes at each end of the link. The switch uses a proprietary algorithm number with ECP to provide STAR key management and DES encryption.

If a PPP interface is configured for compression and encryption, the data is compressed first to give the compression phase the best opportunity of finding non-random sequences, and then the data is encrypted. PPP fully supports the negotiation and link management required to provide both compression and encryption on links. Configuring PPP for dual mode operation requires that both encryption and compression be enabled on the PPP interface. The compression option is negotiated by the switches at each end of the link so that if one is configured for dual mode and the other is configured for encryption, then compression is not used. Encryption on a link is not negotiable—both ends must be configured for encryption for the link to be established.

Command Reference

This section describes the commands available on the switch to configure and monitor the compression and encryption processes on the switch.

A user must be logged in with security officer privilege to configure encryption services. See Chapter 29, User Authentication for more information about creating users with security officer privilege, configuring remote security officers, and logging in with security officer privilege.

For each interface over which compression or encryption is to be used, a higher layer protocol must be configured to use compression or encryption. Currently, compression and encryption are supported on Point-to-Point Protocol (PPP) and Security Associations. See Chapter 11, Point-to-Point Protocol (PPP) for details of the commands required to enable compression and encryption on a PPP interface. See Chapter 13, Internet Protocol (IP) for details of the commands required to enable IP payload compression and encryption with Security Associations. See Chapter 33, Secure Shell for more information about configuring and using Secure Shell.

See "Conventions" on page xlix of About this Software Reference in the front of this manual for details of the conventions used to describe command syntax. See Appendix A, Messages for a complete list of messages and their meanings.

create enco key

Syntax

```
CREate ENCo KEY=key-id TYPe={DES|3DES2key|3DESInner|
   GENeral|RSA} [DESCription=description] [FILe=filename]
   [FORmat={HEX|NIQ|SSH}] [IPaddress=ipadd]
   [LENgth=length] [MODule=module-id] [{RANDom|
   VALue=value}]
```

where:

- *key-id* is a number from 0 to 65535.
- *description* is a character string from 1 to 24 characters long. Valid characters are any printable character. If *description* contains spaces it must be in double quotes.
- *filename* is a valid switch filename with a `.key` extension.
- *ipadd* is an IP address in dotted decimal notation.
- *length* is a number from 0 to 65535.
- *module-id* is the name or number of a switch module (see "Module Identifiers and Names" on page B-2 of Appendix B, Reference Tables for a complete list).
- *value* is a character string, of variable length depending on the key type. For ASCII formatted keys, valid characters are lowercase letters (a–z) and digits (2–9). The digits 0 and 1 are illegal to prevent confusion with the letters O and l. Hexadecimal keys must start with "0x" and must contain an even number of characters using the digits (0-9) and lettersn (a-f). Passphrase keys may contain any printable character. If *value* contains spaces, it must be in double quotes.

Description This command creates a encryption or authentication key of the specified type and stores the key information in the switch's flash memory. This command can also be used to import or export RSA keys.

This command requires a user with security officer privilege when the switch is in security mode. Keys created on a switch that is not operating in security mode are destroyed when the switch is restarted. Enable security mode before creating encryption keys.

Before creating keys, you must first define a user with security officer privileges if you have not already done so. Defining a security officer allows someone to enable security mode when logging in. The security officer definition process must be done on all switches that use the keys—the head office switch and the remote office switch.

The **key** parameter specifies the identification number for the key.

The **type** parameter specifies the type of key to be created. If a DES or 3DES key is being created, then the **random** or **value** parameters must be specified. If **des** is specified, a 56-bit DES key is created. If **3des2key** is specified, a 112-bit DES key is created. If **3desinner** is specified, a 168-bit DES key is created. If an RSA key is being generated, then the **length** or **file** parameters must be specified. If the **file** parameter is specified, the RSA key is imported from or exported to the specified file. If the **file** parameter is not specified, then a random RSA key is generated. If a **general** key is being created, then the **length** or **value** parameters must be specified. **General** keys can be used for authentication algorithms or as shared secrets.

The **description** parameter specifies a user-defined description for the key, to make it easier to keep track of different keys.

The **file** parameter specifies name of a switch file. RSA public keys may be imported from or exported to a file in either Secure Shell format, the switch's own format or in hexadecimal format. If the file exists but the specified RSA key does not exist, then the RSA key is imported from the file. If the specified RSA key does exist but the file does not exist, the RSA key is exported to the file. The **format** parameter must be specified when importing or exporting keys.

The **format** parameter specifies the format of the .key file when importing or exporting an RSA key. Secure Shell users should use SSH. NIQ is the switch's own format, which can be used for transferring RSA keys between switches. The HEX format should be used when transferring keys between other devices. The default is HEX. If **format** is specified, the **file** parameter must also be present.

The **ipaddress** parameter specifies an IP address to associate with the key. The SSH protocol uses this value to find the RSA public key of a remote peer. Documentation for individual protocols specify whether this parameter is required.

The **length** parameter specifies the length of the key to be created. An RSA key length is specified in bits and must be a multiple of 32. Valid RSA keys are from 256 to 2048 bits. When creating a MD5 key, you must specify a length of 16. When creating a SHA key you must specify a length of 20. **General** type keys can have a length from 2 to 64 bytes.

The **module** parameter can be used to link a key to a specific module. Documentation for individual protocols specify whether this parameter is required.

The **random** parameter creates a random key. The key can entered into another switch using the **create enco key** command and specifying the **value** parameter.

The **value** parameter creates a key with the supplied value. DES and 3DES keys require a key in 5-bit ASCII format or hexadecimal format. This ASCII representation includes a check value of the key to ensure it has been typed in correctly. A hexadecimal key always starts with "0x". The value of a key can be displayed using the **show enco key**. **General** type keys can be entered as a string or in hexadecimal format.

Examples To create a random DES encryption key with the identification number 1 and then display the key, use the commands:

```
cre enc key=1 typ=des rand
show enc key=1
```

To add this DES key to another router, use the following command on the other device:

```
cre enc key=1 typ=des val=value
```

where *value* is the value of the key displayed in the output of the **show enco key** command on page 31-45.

To create a random 512-bit RSA private key with the key identification number 2, use the command:

```
cre enc key=2 typ=RSA len=512 desc="Switch A private key"
```

To create an uploadable file for the public component of the same RSA key in the format used by Secure Shell use the command:

```
cre enc key=2 typ=rsa fil=routerA.key for=ssh
```

To import an RSA key from the file `RSA.KEY`, which is in HEX format, as encryption key 3, use the command:

```
cre enc key=3 typ=rsa fil=rsa.key for=hex
```

Related Commands

destroy enco key
set enco key
show enco key

destroy enco key

Syntax `DESTroy ENCo KEY=`*`key-id`* `LOCation=FLAsh`

where *key-id* is a number from 0 to 65535

Description This command destroys the specified encryption key. The memory the key occupied is overwritten to ensure the key is irretrievable. This command requires a user with security officer privilege when the switch is in security mode.

The **key** parameter specifies the identification number for the key. A key with the specified identification number must exist.

The **location** parameter specifies the location of the key.

Examples To destroy the encryption key in flash with the key identification number 4, use the command:

```
dest enc key=4 loc=fla
```

Related Commands
create enco key
set enco key
show enco key

disable enco compstatistics

Syntax `DISable ENCo COMPSTatistics`

Description This command disables the calculation and storage of compression ratio statistics for any compression-only ENCO channels.

Related Commands
enable enco compstatistics
show enco channel

disable enco debugging

Syntax `DISable ENCo DEBugging={CHannel|PAcket|TImestamp|ALL}`

Description This command disables the specified debugging option. The option must currently be enabled. Any combination of options can be disabled by using successive commands.

The **debug** parameter specifies which debugging option to disable. The **packet** parameter disables the debugging of packet contents ENCO processed. The **timestamp** parameter disables the display of encryption and compression operations. The **channel** option is no longer supported.

Examples To disable the debugging of the contents of packets processed, use the command:

```
dis enc deb=pa
```

Related Commands **enable enco debugging**
show enco debug

enable enco compstatistics

Syntax `ENAble ENCo COMPSTatistics`

Description This command enables the calculation and storage of compression ratio statistics for any compression-only ENCO channels. The collected statistics are displayed in the output of the **show enco channel** command on page 31-26.

Related Commands **disable enco compstatistics**

enable enco debugging

Syntax `ENAble ENCo DEBugging={CHannel|PAcket|TImestamp|ALL}`

Description This command enables specific debugging. The specified debugging option must currently be disabled. Debugging information is sent to the terminal where the command was entered. Any combination of options can be enabled using successive commands.

The **debug** parameter specifies the debugging option to enable. Specify **packet** to enable debugging of the contents of packets processed by the ENCO module. Specify **timestamp** to enable measurements of encryption and compression operations. Specify **all** to enable all debugging options. The **channel** option is no longer supported.

Examples To enable the debugging of the contents of packets processed by ENCO, use the command:

```
ena enc deb=pa
```

Related Commands **disable enco debugging**
show enco debug

reset enco counters

Syntax

```
RESET ENCo COUnters={DEs|DH|HMac|PAC|PRed|QUeues|RSa|SSl|
   STac|USer|UTil}
```

Description

This command clears general and process-specific counters. It requires a user with security officer privilege when the switch is in security mode.

The **counters** parameter specifies the category of counters to be cleared. If a category is not specified, all ENCO counters are cleared. The **user**, **util**, and **queues** counters resets information about the general ENCO operation, while the other parameters reset information for particular processes.

- If **des** is specified, counters for the DES encryption process are reset.
- If **dh** is specified, counters are reset for the Diffie-Hellman key exchange process.
- If **hmac** is specified, counters for the HMAC message process are reset.
- If **pred** is specified, counters for the Predictor compression process are reset.
- If **queues** is specified, counters are reset for the internal queues.
- If **rsa** is specified, counters for the RSA encryption process are reset.
- If **ssl** is specified, counters for the SSL process are reset.
- If **stac** is specified, counters are reset for the STAC compression process.
- If **user** is specified, counters are reset for the interface between ENCO and user applications that use ENCO channels.
- If **util** is specified, counters are reset for the interface between ENCO and user applications that use ENCO for one-off jobs.

Examples

To reset all ENCO counters, use the command:

```
reset enc cou
```

To reset all the counters for the Predictor compression process, use the command:

```
reset enc cou=pr
```

Related Commands

show enco counters

set enco dhpadding

Syntax `SET ENCo DHPAdding={ON|OFf}`

Description This command is used to control the padding process for Diffie-Hellman generated values. This may be required when interoperability is required with other vendor's equipment that uses the Diffie-Hellman algorithm.

The **dhpadding** parameter specifies whether values generated by the Diffie-Hellman process should be padded. If on, leading zeroes are inserted; if off, they are not. The default is on.

Examples To turn off Diffie-Hellman padding, use the command:

```
set enco dhpa=of
```

Related Commands **show enco**

set enco dhpriority

Syntax `SET ENCo DHPRiority={High|Medium|Low}`

Description This command is used to change the priority of the Diffie-Hellman key exchange on the switch. The higher the priority, the more CPU time the algorithm uses and the faster the key exchange is completed. If the speed of the Diffie-Hellman key exchange is not critical, the **dhpriority** can be set lower so that the routing process is given more CPU time.

The **dhpriority** parameter specifies the priority of the Diffie-Hellman key exchange. Valid values are **high**, **medium**, and **low** and the default is **high**.

Examples To set the Diffie-Hellman priority to low, use the command:

```
set enc dhpr=l
```

Related Commands **show enco key**

set enco key

Syntax

```
SET ENCo KEY=key-id [DESCription=description]
   [IPaddress=ipadd] [MODule=module-id]
```

where:

- *key-id* is a number from 0 to 65535.
- *description* is a character string 1 to 24 characters long. Valid characters are any printable character. If *description* contains spaces, it must be in double quotes.
- *ipadd* is an IP address in dotted decimal notation.
- *module-id* is the name or number of a switch module (see "Module Identifiers and Names" on page B-2 of Appendix B, Reference Tables for a complete list).

Description

This command changes the user-defined description, IP address or module for a specific key. It requires a user with security officer privilege when the switch is in security mode.

The **key** parameter specifies the identification number for the key. The specified encryption key must already exist.

The **description** parameter specifies a user-defined description for the key, to make it easier to keep track of different keys.

The **ipaddress** parameter specifies an IP address to associate with the key. The SSH protocol uses this value to find the RSA public key of a remote peer. Documentation for particular modules specify whether this parameter is required.

The **module** parameter can be used to link a key to a specific module. Documentation for individual protocols specify whether this parameter is required.

Examples

To change the description for key 1, use the command:

```
set enc key=1 desc="Switch Z key"
```

Related Commands

create enco key
destroy enco key
show enco key

set enco sw

Syntax `SET ENCo SW [PREDChannels=0..4] [STACChannels=0..4]`

Description This command changes the configuration parameters for software compression.

The **predchannels** parameter specifies the number of Predictor compression channels to allocate. Each Predictor compression channel requires 128 KBytes of contiguous memory. On a switch with 4MB of memory or less, the number of Predictor channels is limited to 2.

The **stacchannels** parameter specifies the number of STAC LZS compression channels to allocate. Each STAC LZS compression channel requires 13 KBytes of contiguous memory.

Examples To configure three STAC LZS software compression channels, use the command:

```
set enc sw stacc=3
```

Related Commands **show enco channel**

show enco

Syntax `SHow ENCo`

Description This command displays information about ENCO. The output of this command varies depending on feature licences (Figure 31-4, Table 31-1).

Figure 31-4: Example output from the **show enco** command

```
ENCO Module Configuration:
  Hardware ............................... NOT PRESENT
  Lowest valid channel ................... 1
  Highest valid channel .................. 2047
  Compression Statistics ................. DISABLED
  Diffie Hellman Priority ................ HIGH
  Diffie Hellman Padding ................. ON

  SW Processes available
    STAC - Stac Compression
    DES  - DES Encryption for Secure Management
    3DES - Triple DES Encryption
    RSA  - RSA Encryption
    DH   - Diffie Hellman
    SSL  - Secure Socket Layer
    HMAC - Message Digest

  Stac LZS compression performance level . 3
  Stac LZS compression footprint ......... 9989
  Stac LZS compression history size ...... 20688
  Stac LZS decompression history size .... 4168
  Stac LZS channels configured ........... 2
  Stac LZS channels available ............ 2
```

Table 31-1: Parameters in output of the **show enco** command

Parameter	Meaning
Hardware	Hardware encryption is not supported on the switch.
Lowest valid channel	Identification number of the lowest channel available for use by a user application.
Highest valid channel	Identification number of the highest channel available for use by a user application.
Compression Statistics Enabled	Whether gathering compression statistics is enabled for all channels.
Diffie Hellman Priority	Whether the priority of the Diffie-Hellman key exchange is high, medium, or low.
Diffie Hellman Padding	Whether values that Diffie-Hellman generates are padded.

Table 31-1: Parameters in output of the **show enco** command (cont)

Parameter	Meaning
SW Processes available	A list of the software-based processes available to ENCO for processing user data packets: NONE DMAN PREDICTOR STAC RSA DH SSL HMAC DES
STAC LZS information displayed if you configured STAC channels with the set enco sw command	
Stac LZS compression performance level	The performance level between 0 (maximum compression ratio) and 3 (maximum compression speed).
Stac LZS compression footprint	Value in bytes.
Stac LZS compression history size	Value in bytes.
Stac LZS decompression history size	Value in bytes.
Stac LZS channels configured	Number of configured channels.
Stac LZS channels available	Number of available channels.

Related Commands

set enco dhpriority
set enco dhpadding
show enco channel
show enco counters

show enco channel

Syntax `SHow ENCo CHannel[=`*channel* `[COUnters]]`

where *channel* is a number from 0 to 512, if the switch has up to 8 Mbytes of RAM, 0 to 1024 if the switch has 16 Mbytes of RAM, or 0 to 2048 if the switch has 32 Mbytes of RAM

Description This command displays information about active ENCO channels. If an ENCO channel is not specified, a summary of all currently active channels is displayed (Figure 31-5, Table 31-2). If an ENCO channel is specified, detailed configuration and status information about the specified channel is displayed (Figure 31-6 on page 31-27, Table 31-3 on page 31-27). If compression statistics are enabled, the display includes compression statistics.

If the **counters** parameter is specified, information counters for the specified channel are displayed (Figure 31-7 on page 31-29, Table 31-4 on page 31-29).

Figure 31-5: Example output from the **show enco channel** command

```
Channel  State    User UserID   MDL   pktOverhead Process
-------------------------------------------------------------
   1       UP       SA f0000001 1528    72         DES
   2       UP      PPP 00000001 1500    64         DES
   3       UP      SSH 00000001 1584    16         DES
-------------------------------------------------------------
```

Table 31-2: Parameters in output of the **show enco channel** command

Parameter	Meaning
Channel	Channel identification number.
State	Whether the channel is up or down.
User	The user application attached to this channel: PPP FR MIOX TEST SA SSH HTTP LOADBAL
UserID	Number used by the user application to identify this channel.
MDL	Maximum data length of packets accepted on this channel.
pktOverhead	Number of bytes that the user application requested be reserved in a packet in front of encoded data.
Process	Process for which the channel is configured: PREDICTOR SSL STAC RSA DH DES HMAC

Figure 31-6: Example output from the **show enco channel** command for a specific channel

```
Channel ..................... 1

Type ........................ ENCODE/DECODE
State ....................... UP
User ........................ SSH
User ID ..................... 00000001
Maximum Data Length ......... 1584
Packet Overhead ............. 16
Process ..................... DES
Process Configuration:
  Des Type..........DES - 56 bit
  Check Type .......NONE
  Channel Type......ENCODE/DECODE
  History Mode......Off
  IV Type...........Random
```

Table 31-3: Parameters in output of the **show enco channel** command for a specific channel

Parameter	Meaning
Channel	Identification number of the channel.
Type	Mode of the channel: ENCODE/DECODE ENCODE ONLY DECODE ONLY
State	Whether the channel is up or down.
User	User application attached to this channel: PPP FR MIOX TEST SA SSH HTTP LOADBAL
User ID	Number used by the user application to identify this channel.
Maximum Data Length	Maximum data length of packets accepted on this channel.
Packet Overhead	Number of bytes reserved at the head of data packets in front of the encoded data, for lower layer packet headers.
Process	Process for which the channel is configured: RSA DH DES PREDICTOR SSL STAC
Process Configuration	Details about a particular process. The fields displayed vary depending on the process.

Table 31-3: Parameters in output of the **show enco channel** command for a specific channel (cont)

Parameter	Meaning
Max Data Length	Maximum allowed length of data packets on the channel.
Check Type	Type of checksum to be used: XOR8 NONE (STAC compression) CRCCCITT CRC16(Predictor compression)
DES Type	[DES] The DES encryption/decryption algorithm used to process packets on the channel: DES–56 bit 3DES–112 bit–outer CBC 3DES–168 bit–inner CBC 3DES–168 bit–outer CBC
Channel Type	[DES] The mode of the channel: ENCODE/DECODE ENCODE ONLY DECODE ONLY
History Mode	[DES] Whether the process is operating with history mode enabled; one of On or Off.
IV Type	[DES] The type of Initialisation Vector (IV) used; one of Zero, Random, or Specified.
RSA mode	[RSA] Whether the RSA encryption mode on this channel is public or private.
Mode	[Diffie-Hellman] Whether the mode is Phase 1 or Phase 2.
Group Type	[Diffie-Hellman] The group types supported. Only MODP is currently supported.
Group	[Diffie-Hellman] Whether the Diffie-Hellman group is 768-bit MODP or 1024-bit MODP.
Algorithm	[HMAC] Whether the HMAC algorithm is MD5 or SHA.
Key Length	[HMAC] The length of the HMAC key.
Compression Statistics	Statistics for the compression process. This section is only displayed when compression statistics have been enabled with the **enable enco compstatistics**.
Number of Packets Compressed	Number of data packets that have been compressed.
Best Compression Ratio	Highest compression ratio achieved.
Mean Compression Ratio	Mean compression ratio achieved.
Worst Compression Ratio	Lowest compression ratio achieved.
Compression Ratio	Range of compression ratios.
Number of Packets	Number of packets compressed, for which the resulting compression ration was in the specified range.

Figure 31-7: Example output from the **show enco channel counter** command

```
Channel Counter:

 UP events                1  DOWN events              0
 start config             1  attach good              1
 encode NULL packets      0  decode NULL packets      0
 enc bad priorities       0  dec bad priorities       0
 encode bad length        0  decode bad length        0
 encode actions sent      0  decode actions sent      0
 good encodes             0  good decodes             0
 bad encodes              0  bad decodes              0
 reset E actions sent     0  reset D actions sent     0
 good encode resets       0  good decode resets       0
 bad encode resets        0  bad decode resets        0
 discarded enc jobs       0  discarded dec jobs       0
```

Table 31-4: Parameters in output of the **show enco channel counters** command

Parameter	Meaning
UP events	Number of times the channel has entered the up state.
DOWN events	Number of times the channel has entered the down state.
start config	Number of times a configure operation has started on the channel.
attach good	Number of successful attach operations on the channel.
encode NULL packets	Number of encode requests received from a user application with no data packet.
decode NULL packets	Number of decode requests received from a user application with no data packet.
encode bad priorities	Number of encode requests received from a user application with a data packet containing an unknown priority.
decode bad priorities	Number of decode requests received from a user application with a data packet containing an unknown priority.
encode bad length	Number of encode requests received from a user application with a data packet with a bad length.
decode bad length	Number of decode requests received from a user application with a data packet with a bad length.
encode actions sent	Number of encode actions that have been sent to the process on this channel.
decode actions sent	Number of decode actions have been sent to the process on this channel.
good encodes	Number of successful encode operations on the channel.
good decodes	Number of successful decode operations on the channel.
bad encodes	Number of unsuccessful encode operations on the channel.
bad decodes	Number of unsuccessful decode operations on the channel.
reset E actions sent	Number of encode reset actions that have been sent to the process on the channel.
reset D actions sent	Number of decode reset actions that have been sent to the process on the channel.

Table 31-4: Parameters in output of the **show enco channel counters** command (cont)

Parameter	Meaning
good encode resets	Number of successful encode resets on the channel.
good decode resets	Number of successful decode resets on the channel.
bad encode resets	Number of unsuccessful encode resets on the channel.
bad decode resets	Number of unsuccessful decode resets on the channel.
discarded encode jobs	Number of encode jobs discarded due to queue overloading or a channel reset.
discarded decode jobs	Number of decode jobs discarded due to queue overloading or a channel reset.

Examples

To show a summary of all active ENCO channels, use the command:

```
sh enc ch
```

To show detailed configuration and status information for channel 1, use the command:

```
sh enc ch=1
```

To show counter information for channel 1, use the command:

```
sh enc ch=1 cou
```

Related Commands

show enco counters

show enco counters

Syntax

```
SHow ENCo COUnters={DEs|DH|HMac|PAC|PRED|QUeues|RSa|SSl|
   STac|USer|UTil}
```

Description

This command displays counter information about ENCO.

The **counters** parameter specifies the category of counters to display. The **user**, **util**, and **queues** counters display information about the general ENCO operation, while the other parameters display information for particular processes.

- If **des** is specified, counters for the DES encryption process are displayed (Figure 31-8 on page 31-32, Table 31-5 on page 31-32).
- If **dh** is specified, counters for the Diffie-Hellman key exchange process are displayed (Figure 31-9 on page 31-34, Table 31-6 on page 31-34).
- If **hmac** is specified, counters for the HMAC message process are displayed (Figure 31-10 on page 31-34, Table 31-7 on page 31-34).
- If **pred** is specified, counters for the PRED compression process are displayed (Figure 31-12 on page 31-35, Table 31-9 on page 31-35).
- If **queues** is specified, counters are displayed for the internal queues of ENCO (Figure 31-11 on page 31-35, Table 31-8 on page 31-35).
- If **rsa** is specified, counters for the RSA encryption process are displayed (Figure 31-13 on page 31-36, Table 31-10 on page 31-36).
- If **ssl** is specified, counters for the SSL process are displayed (Figure 31-14 on page 31-36, Table 31-11 on page 31-37).
- If **stac** is specified, counters for the STAC compression process are displayed (Figure 31-15 on page 31-39, Table 31-12 on page 31-39).
- If **user** is specified, counters are displayed for the interface between ENCO and user applications that use ENCO channels (Figure 31-16 on page 31-42, Table 31-13 on page 31-42).
- If **util** is specified, counters are displayed for the interface between ENCO and user applications that use ENCO channels for one-off jobs (Figure 31-17 on page 31-43, Table 31-14 on page 31-44).

Figure 31-8: Example output from the **show enco counter=des** command

```
ENCO Process DES/3DES Counter:

  configGood              1   configBad               0
  configNoResource        0   configNotSSH            0
  BadBuffer               0   BadAlign                0
  BadLength               0   nohistory               0
  desJobs                 0   3Des2KeyJobs            0
  3DesInnerJobs           0   noHistJobs              0
  desMacJobs              0
  badDesType              0   badJobType              0

  unknownJob              0   error                   0
  reset                   0   confNotDes              0
  commWaitTimeOut         0   dataInWaitTimeOut       0
  dataOutWaitTimeOut      0

  goodDecrypt             0   goodEncrypt             0
  badDecrypt              0   badEncrypt              0

  DMA1Start               0   DMA2Start               0
  DMA1Done                0   DMA2Done                0
  DMABed                  0   DMABes                  0
  DMABrkp                 0   DMAConf                 0
  DMA1TimeOut             0   DMA2TimeOut             0
```

Table 31-5: Parameters in output of the **show enco counter=des** command

Parameter	Meaning
configGood	Number of successful channel configurations.
configBad	Number of unsuccessful configuration attempts.
configNoResource	Number of configure attempts without resources.
configNotSSH	Number of attempts to configure a software DES channel when the user was not Secure Shell.
badBuffer	Number of jobs received by the DES/3DES encryption algorithm unit with a bad buffer.
badAlign	Number of jobs received by the DES/3DES encryption algorithm unit with a bad alignment of the packet.
badLength	Number of jobs received by the DES/3DES encryption algorithm unit with a bad length (not a multiple of the DES block length).
nohistory	Number of jobs received by the DES/3DES encryption algorithm unit without valid history (IV's).
desJobs	Number of 56-bit DES jobs received by the DES/3DES algorithm unit.
3Des2KeyJobs	Number of 112-bit 3DES jobs received by the DES/3DES algorithm unit.
3DesInnerJobs	Number of 168-bit 3DES jobs received by the DES/3DES algorithm unit.
noHistJobs	Number of jobs processed by the DES/3DES encryption algorithm unit with history mode set to OFF.
desMacJobs	Number of DES-MAC authentication jobs received by the DES/3DES algorithm unit.

Table 31-5: Parameters in output of the **show enco counter=des** command (cont)

Parameter	Meaning
badDesType	Number of jobs received by the encryption algorithm unit with a invalid DES type.
badJobType	Number of jobs received by the DES/3DES encryption algorithm unit with an invalid job type.
unknownJob	Number of unknown jobs received by the DES/3DES encryption algorithm unit.
error	Number of errors that occurred in the DES/3DES encryption algorithm unit while processing data.
reset	Number of resets by the hardware encryption unit.
confNotDes	Number of attempts to configure a DES channel with an invalid encryption type.
commWaitTimeOut	Number of commands entered for the hardware encryption unit before it was ready for the new command.
dataInWaitTimeOut	Number of times the data was entered to the hardware encryption unit before it is ready for new data.
dataOutWaitTimeOut	Number of times data was read from the hardware encryption unit before it was ready to output new data.
goodDecrypt	Number of good decryption jobs processed by the DES/ 3DES algorithm unit.
goodEncrypt	Number of good encryption jobs processed by the DES/ 3DES algorithm unit.
badDecrypt	Number of bad decryption jobs processed by the DES/3DES algorithm unit.
badEncrypt	Number of bad encryption jobs processed by the DES/3DES algorithm unit.
DMA1Start	Number of times the DMA1 channel started.
DMA2Start	Number of times the DMA2 channel started.
DMA1Done	Number of times the DMA1 channel completed a transfer.
DMA2Done	Number of times the DMA2 channel completed a transfer.
DMABed	Number of times the Bus Error Destination occurred during DMA transfers.
DMABes	Number of times a Bus Error Source occurred during DMA transfers.
DMAbrkp	Number of times a DMA break point interrupt occurred.
DMAConf	Number of times a DMA configuration error occurred.
DMA1TimeOut	Number of times the DMA1 channel timeout occurred.
DMA2TimeOut	Number of times the DMA2 channel timeout occurred.

Figure 31-9: Example output from the **show enco counter=dh** command

```
ENCO Process Diffie-Hellman Counter:

  goodPhase1                1    badPhase1                 0
  goodPhase2                1    badPhase2                 0
  goodConfigure             2    badConfigure              0
  badGroupType              0    badGroup                  0
  badGroupParameters        0    badDataLength             0
  noResources               0    unknownJob                0
```

Table 31-6: Parameters in output of the **show enco counter=dh** command

Parameter	Meaning
goodPhase1	Number of good jobs for phase 1 of the D-H exchange.
badPhase1	Number of failed jobs for phase 1 of the D-H exchange.
goodPhase2	Number of good jobs for phase 2 of the D-H exchange.
badPhase2	Number of failed jobs for phase 2 of the D-H exchange.
goodConfigure	Number of good channel configurations.
badConfigure	Number of failed channel configurations.
badGroupType	Number of jobs with an invalid Group Type.
badGroup	Number of jobs with an invalid Group.
badGroupParameters	Number of jobs with invalid group parameters.
badDataLength	Number of jobs with a bad data length.
noResources	Number of configure jobs with no resources.
unknownJob	Number of unknown jobs.

Figure 31-10: Example output from the **show enco counter=hmac** command

```
ENCO Process MD5 Counter:

  goodHashMD5               1    badHashMD5                0
  goodHashSHA               0    badHashSHA                0
  goodConfigure             1    badConfigure              0
  badAlgorithm              0    noResources               0
  badKeyLength              0    unknownJob                0
  badDataLength             0
```

Table 31-7: Parameters in output of the **show enco counter=hmac** command

Parameter	Meaning
goodHashMD5	Number of good MD5 hashes.
badHashMd5	Number of failed MD5 hashes.
goodHashSHA	Number of good SHA hashes.
badHashSHA	Number of failed SHA hashes.
goodConfigure	Number of good channel configurations.
badConfigure	Number of failed channel configurations.
badAlgorithm	Number of channel configurations with invalid algorithm types.

Table 31-7: Parameters in output of the **show enco counter=hmac** command (cont)

Parameter	Meaning
noResources	Number of channel configurations with no resources
badKeyLength	Number of jobs with an invalid key length.
unknownJob	Number of unknown jobs.
badDataLength	Number of jobs with an invalid data length.

Figure 31-11: Example output from the **show enco counter=queues** command

```
ENCO Queues                        Queued  Discarded  Processed
   Immediate Input queue........      0          0          0
   Priority 0 Input queue (high)      0          0          0
   Priority 1 Input queue ......      0          0          0
   Priority 2 Input queue ......      0          0          0
   Priority 3 Input queue ......      0          0          0
   Priority 4 Input queue ......      0          0          0
   Priority 5 Input queue ......      0          0          0
   Priority 6 Input queue ......      0          0          0
   Priority 7 Input queue ......      0          0          0
   Priority 8 Input queue (low).      0          0          0
   Output queue.................      0          0          0

Input Queue Length Limit........         250
Lowest Input Priority Queue.....           0
Highest Input Priority Queue....           0
```

Table 31-8: Parameters in output of the **show enco counter=queues** command

Parameter	Meaning
ENCO Queues	Internal queues.
Immediate Input queue	Queue for jobs required to be done immediately.
Priority *n* Input queue	Prioritized input queue.
Output queue	Output queue.
Queued	Current number of jobs in the queue.
Discarded	Number of jobs discarded from the queue.
Processed	Number of jobs processed from the queue.
Input Queue Length Limit	Maximum total length of the input queue.
Lowest Input Priority Queue	Lowest input priority queue with queued actions.
Highest Input Priority Queue	Highest priority queue with queued actions.

Figure 31-12: Example output from the **show enco counter=pred** command

```
predictorResets                  0
```

Table 31-9: Parameter in output of the **show enco counter=pred** command

Parameter	Meaning
predictorResets	Number of times the Predictor compression history has been reset.

Figure 31-13: Example output from the **show enco counter=rsa** command

```
ENCO Process RSA Counter:

  goodPublicEncrypt         0   badPublicEncrypt          0
  goodPrivateDecrypt        1   badPrivateDecrypt         0
  goodPrivateEncrypt        0   badPrivateEncrypt         0
  goodPublicDecrypt         0   badPublicDecrypt          0
  goodGenerateKey           0   badGenerateKey            0
  badDataLength             0   badKey                    0
```

Table 31-10: Parameters in output of the **show enco counter=rsa** command

Parameter	Meaning
goodPublicEncrypt	Number of encryption jobs using an RSA public key.
goodPrivateDecrypt	Number of decryption jobs using an RSA private key.
goodPrivateEncrypt	Number of encryption jobs using an RSA private key.
goodPublicDecrypt	Number of decryption jobs using an RSA public key.
goodGenerateKey	Number of RSA keys that have been generated.
badDataLength	Number of jobs with a bad data length.
badPublicEncrypt	Number of failed encryption jobs using an RSA public key.
badPrivateDecrypt	Number of failed decryption jobs using an RSA private key.
badPrivateEncrypt	Number of failed encryption jobs using an RSA private key.
badPublicDecrypt	Number of failed decryption jobs using an RSA public key.
badGenerateKey	Number of failed key generations.
badKey	Number of jobs where the RSA key was invalid.

Figure 31-14: Example output from the **show enco counter=ssl** command

```
ENCO Process SSL Counters:
 initialised ..............  1   initNoResources ...........  0
 configGood ...............  0   configNoConnection ........  0
 configBadUserArgs .........  0   configNoResources .........  0
 destroyGood ..............  0   destroyNoConnection .......  0
 unknownJob ...............  0   doJobNoConnection .........  0

 Application Data:
   appDataEncoded ..........  0     appDataEncodeFail .......  0
   appDataHmacEncFail ......  0     appDataDesEncFail .......  0
   appDataDecoded ..........  0     appDataDecodeFail .......  0
   appDataDesDecFail .......  0     appDataHmacDecFail ......  0

 Handshake:
   genMasterSecrtGood ......  0     genKeyMaterialGood ......  0
   ccsGood .................  0     ccsFail .................  0
   ccsDesConfigFail ........  0     ccsHmacConfigFail .......  0
   processSKEGood ..........  0     processSKENoKey .........  0
   procsSKECfgRSAFail ......  0     procsSKERSADecFail ......  0
   genCKEGood ..............  0     genCKENoKey .............  0
   genCKEConfgRSAFail ......  0     genCKERSAEncFail ........  0
   processCKEGood ..........  0     processCKENoKey .........  0
   procsCKECfgRSAFail ......  0     procsCKERSADecFail ......  0
   genCVGood ...............  0     genCVNoKey ..............  0
   genCVConfigRSAFail ......  0     genCVRSAEncodeFail ......  0
   processCVGood ...........  0     processCVNoKey ..........  0
   procssCVCfgRSAFail ......  0     procssCVRSADecFail ......  0
    processCVFail ..........  0
```

Table 31-11: Parameters in output of the **show enco counter=ssl** command

Parameter	Meaning
initialized	Number of Initialization operations performed.
initNoResources	Number of Initialization attempts without resources.
configGood	Number of successful channel configurations.
configNoConnection	Number of unsuccessful configuration attempts.
configBadUserArgs	Number of unsuccessful configuration attempts due to bad user arguments.
configNoResources	Number of unsuccessful configuration attempts due to lack of resources.
destroyGood	Number of successful channel configuration de-allocations.
destroyNoConnection	Number of channel configuration de-allocation attempts with no connection.
unknownJob	Number of unknown jobs received.
doJobNoConnection	Number of jobs received with an invalid connection.
appDataEncoded	Number of successful attempts to encode application data.
appDataEncodeFail	Number of unsuccessful attempts to encode application data.
appDataHmacEncFail	Number of unsuccessful attempts to hash application data.
appDataDesEncFail	Number of unsuccessful attempts to encrypt application data.
appDataDecoded	Number of successful attempts to decode application data.
appDataDecodeFail	Number of unsuccessful attempts to decode application data.
appDataDesDecFail	Number of unsuccessful attempts to decrypt application data.
appDataHmacDecFail	Number of unsuccessful attempts to authenticate application data.
genMasterSecrtGood	Number of successful attempts to generate the Master Secret.
genKeyMaterialGood	Number of successful attempts to generate the key material.
ccsGood	Number of successful Change Cipher Spec's processed.
ccsFail	Number of unsuccessful Change Cipher Spec's.
ccsDesConfigFail	Number of unsuccessful Change Cipher Spec's due to failed DES configuration.
ccsHmacConfigFail	Number of unsuccessful Change Cipher Spec's due to failed HMAC configuration.
processSKEGood	Number of successful Server Key Exchange messages processed.
processSKENoKey	Number of failed SKE messages due to no key.
procsSKECfgRSAFail	Number of failed SKE messages due to failed RSA configuration.
procsSKERSADecFail	Number of failed SKE messages due to failed RSA decryption.
genCKEGood	Number of successful Client Key Exchange messages generated.

Table 31-11: Parameters in output of the **show enco counter=ssl** command (cont)

Parameter	Meaning
genCKENoKey	Number of unsuccessful CKE message generations due to no key.
genCKEConfgRSAFail	Number of unsuccessful CKE message generations due to failed RSA configuration.
genCKERSAEncFail	Number of unsuccessful CKE message generations due to failed RSA encryption.
processCKEGood	Number of successful Client Key Exchange messages processed.
processCKENoKey	Number of failed CKE messages due to no key.
procsCKECfgRSAFail	Number of failed CKE messages due to failed RSA configuration.
procsCKERSADecFail	Number of failed CKE messages due to failed RSA decryption.
genCVGood	Number of successful Certificate Verify messages generated.
genCVNoKey	Number of unsuccessful CV message generations due to no key.
genCVConfigRSAFail	Number of unsuccessful CV message generations due to failed RSA configuration.
genCVRSAEncodeFail	The number of unsuccessful CV message generations due to failed RSA encryption.
processCVGood	Number of successful Certificate Verify messages processed.
processCVNoKey	Number of failed CV messages due to no key.
procssCVCfgRSAFail	Number of failed CV messages due to failed RSA configuration.
procssCVRSADecFail	Number of failed CV messages due to failed RSA decryption.
processCVFail	Number of failed CV messages.

Figure 31-15: Example output from the **show enco counter=stac** command

```
General Enco STAC Counters:
  procHandPrmBadJobTyp ..... 0       procHandParmNullCfg ..... 0
  procHandPrmNullInPkt ..... 0       procHandPrmNllOutPkt ..... 0
  procHandParmBadMdl ..... 0
  procHandHwCompFail ..... 0         procHandSwCompFail ..... 0
  procHandCmpNullChkPt ..... 0       procHandCompGood ..... 0
  procHandHwDecompFail ..... 0       procHandSwDecompFail ..... 0
  procHandDecmpChkFail ..... 0       procHandDecompGood ..... 0
  procHandConfEDone ..... 0          procHandConfDDone ..... 0
  procHandResetEDone ..... 0         procHandResetDDone ..... 0
  procHandFailRecnfJob ..... 0       procHandFailUnkwnJob ..... 0
  configNoResource ..... 0           configHwNoHistory ..... 0
  configSwNoHistory ..... 0          configGood ..... 0
  destroyParmNullConfg ..... 0       destroyGood ..... 0

Enco STAC SW Counters:
  compParmNullHistPt ..... 0         compParmNullSourcePt ..... 0
  compParmNullResultPt ..... 0       compParmBadMdl ..... 0
  compMdlAbort ..... 0               compError ..... 0
  compGood ..... 0
  decompParmNullHistPt ..... 0       decompParmNullSourPt ..... 0
  decompParmNullReslPt ..... 0       decompParmBadMdl ..... 0
  decompMdlAbort ..... 0             decompDestExhaust ..... 0
  decompEocMissing ..... 0           decompError ..... 0
  decompGood ..... 0
  resetParmNullHistPt ..... 0
  rstDcmpParmNullHstPt ..... 0       resetDecompGood ..... 0
  resetDecompError ..... 0
  dmyCompParmNullHstPt ..... 0       dummyCompError ..... 0
  dummyCompGood ..... 0
  setSpeedParmBadSpeed ..... 0
  allocHistParmBadChan ..... 0       allocHstChainNotCntg ..... 0
  allocHstChainTooShrt ..... 0       allocHistGood ..... 0
```

Table 31-12: Parameters in output of the **show enco counter=stac** command

Parameter	Meaning
procHandPrmBadJobTyp	Number of times the STAC process handler received a bad job type parameter.
procHandParmNullCfg	Number of times the STAC process handler received a job with a NULL STAC configuration.
procHandPrmNullInPkt	Number of times the STAC process handler received a job with a NULL In Packet.
procHandParmNllOutPkt	Number of times the STAC process handler received a job with a NULL Out Packet.
procHandParmBadMdl	Number of times the STAC process handler received a job with an invalid Maximum Data Length value.
procHandHwCompFail	Number of times the STAC process handler had a failed hardware compression job.
procHandSwCompFail	Number of times the STAC process handler had a failed software compression job.
procHandCompNullCheckPt	Number of times the STAC process handler received a job with a NULL check pointer.
procHandCompGood	Number of times the STAC process handler had a successful compression job.

Table 31-12: Parameters in output of the **show enco counter=stac** command (cont)

Parameter	Meaning
procHandHwDecompFail	Number of times the STAC process handler had a failed hardware decompression job.
procHandSwDecompFail	Number of times the STAC process handler had a failed software decompression job.
procHandDecompCheckFail	Number of times the STAC process handler had a software decompression job with a bad checksum.
procHandDecompGood	Number of times the STAC process handler had a successful decompression job.
procHandConfEDone	Number of times the STAC process handler had a successful compression channel configure job.
procHandConfDDone	Number of times the STAC process handler had a successful decompression channel configure job.
procHandResetEDone	Number of times the STAC process handler had a successful compression channel reset job.
procHandResetDDone	Number of times the STAC process handler had a successful decompression channel reset job.
procHandFailReconfJob	Number of times the STAC process handler had a failed channel reconfigure job.
procHandFailUnknownJob	Number of times the STAC process handler failed because it received an unknown job.
configNoResource	Number of times a STAC channel configure failed because there were no ENCO channels available.
configHwNoHistory	Number of times a STAC channel configure failed because there were no hardware histories available.
configSwNoHistory	Number of times a STAC channel configure failed because there were no software histories available.
configGood	Number of successful STAC channel configure jobs.
destroyParmNullConfig	Number of times a STAC channel destroy failed because the channel did not exist.
destroyGood	Number of successful STAC channel destroy jobs.
compParmNullHistoryPt	Number of times a software compression job was received with a NULL history pointer.
compParmNullSourcePt	Number of times a software compression job was received with a NULL source packet pointer.
compParmNullResultPt	Number of times a software compression job was received with a NULL result packet pointer.
compParmBadMdl	Number of times a software compression job was received with an invalid maximum data length value.
compMdlAbort	Number of times a software compression job failed due to the maximum data length being exceeded.
compError	Number of times a software compression job failed due to an unspecified error.
compGood	Number of successful software compression jobs.
decompParmNullHistoryPt	Number of times a software decompression job was received with a NULL history pointer.

Table 31-12: Parameters in output of the **show enco counter=stac** command (cont)

Parameter	Meaning
decompParmNullSourcePt	Number of times a software decompression job was received with a NULL source packet pointer.
decompParmNullResultPt	Number of times a software decompression job was received with a NULL result packet pointer.
decompParmBadMdl	Number of times a software decompression job was received with an invalid maximum data length value.
decompMdlAbort	Number of times a software decompression job failed due to the maximum data length being exceeded.
decompDestExhuast	Number of times a software decompression job failed due to the result data being exhausted.
decompEocMissing	Number of times a software decompression job failed due to the failure to find an end of compressed data marker.
decompError	Number of times a software decompression job failed due to an unspecified error.
decompGood	Number of successful software decompression jobs.
resetParmNullHistoryPt	Number of times a software compression channel reset job was received with a NULL history pointer.
resetDecompParmNullHistoryPt	Number of times a software decompression channel reset job was received with a NULL history pointer.
resetDecompGood	Number of successful software decompression channel resets.
resetDecompError	Number of failed software decompression channel resets.
dummyCompParmNullHistoryPt	Number of times a software compression channel dummy compression job was received with a NULL history pointer.
dummyCompError	Number of failed software compression channel dummy compressions.
dummyCompGood	Number of successful software compression channel dummy compressions.
setSpeedParmBadSpeed	Number of times a set software compression speed job was received with an invalid speed value.
allocHistParmBadChan	Number of times an allocate software compression history job was received for an invalid channel.
allocHistChainNotContig	Number of times an allocate software compression history job failed due to a non-contiguous memory allocation.
allocHistChainTooShort	Number of times an allocate software compression history job failed because it did not have enough non-contiguous memory.
allocHistGood	Number of successful allocate software compression history jobs.

Figure 31-16: Example output from the **show enco counter=user** command

```
ENCO User Interface Counter:

  startConfig              2    startReconfig           0
  attachGood               2    attachFail              0
  attachNoConfig           0    attachBadUserType       0
  attachedInvalidChan      0    attachedUnusedChan      0
  attachProcNotAvail       0

  reconfigInvalidChan      0    reconfigUnusedChan      0
  reconfigNoConfig         0

  detachInvalidChannel     0    detachUnusedChannel     0
  detachedInvalidChan      0    detachedUnusedChan      0
  detachGood               0

  decodeInvalidChannel     0    decodeUnusedChannel     0
  encodeInvalidChannel     0    encodeUnusedChannel     0
  codedInvalidChannel      0    codedUnusedChannel      0
  resetInvalidChannel      0    resetUnusedChannel      0
  resetDoneInvalidChan     0    resetDoneUnusedChan     0

  configBadMode            0    configBadUserType       0
  configBadPktLength       0    configBadEncrType       0
  configBadCompType        0    configBadHistoryMode    0
  configBadCheckType       0

  discardInvalidChan       0    discardUnusedChannel    0
```

Table 31-13: Parameters in output of the **show enco counter=user** command

Parameter	Meaning
startConfig	Number of channel configuration requests initiated.
startReconfig	Number of channel reconfiguration requests started.
attachGood	Number of successful channel attaches.
attachFail	Number of unsuccessful channel attaches.
attachNoConfig	Number of channel attach requests received with no configuration information.
attachBadUserType	Number of channel attach requests containing a bad user type.
attachedInvalidChannels	Number of channel attached events on invalid channels.
attachedUnusedChannel	Number of channel attached events on unused channels.
attachProcNotAvail	Number of attaches requesting a process while the process is unavailable.
reconfigInvalidChannel	Number of channel reconfigure requests on invalid channels.
reconfigUnusedChannel	Number of channel reconfigure requests on unused channels.
reconfigNoConfig	Number of channel reconfigure requests received with no configuration information.
detachInvalidChannel	Number of channel detach requests on nonexistent channels.
detachedInvalidChannel	Number of channel detached events on invalid channels.

Table 31-13: Parameters in output of the **show enco counter=user** command

Parameter	Meaning
detachedUnusedChannel	Number of channel detached events on unused channels.
detachGood	Number of successful channel detaches.
decodeInvalidChannel	Number of decode requests on nonexistent channels.
decodeUnusedChannel	Number of decode requests on unused channels.
encodeInvalidChannel	Number of encode requests on nonexistent channels.
encodeUnusedChannel	Number of encode requests on unused channels.
codedInvalidChannel	Number of encode events on nonexistent channels.
codedUnusedChannel	Number of encode events on unused channels.
resetInvalidChannel	Number of channel reset requests on nonexistent channels.
resetUnusedChannel	Number of channel reset requests on unused channels.
resetDoneInvalidChannel	Number of channel reset requests on invalid channels.
resetDoneUnusedChannel	Number of channel reset requests on nonexistent channels.
configBadMode	Number of channel configuration requests containing a bad mode.
configBadUserType	Number of channel configuration requests containing a bad user type.
configBadPktLength	Number of channel configuration requests containing a bad packet length.
configBadEncrType	Number of channel configuration requests containing a bad encryption type.
configBadCompType	Number of channel configuration requests containing a bad compression type.
configBadHistoryMode	Number of channel configuration requests containing a bad history mode.
configBadCheckType	Number of channel configuration requests containing a check type.
discardInvalidChannel	Number of discarded jobs on invalid channels.
discardUnusedChannel	Number of discarded jobs on nonexistent channels.

Figure 31-17: Example output from the **show enco counter=util** command

```
ENCO Utility Counter:

  codeNullPacket          0    codeBadPacketPriorit    0
  codeBadPacketLength     0    codeBadConfig           0
  actionSentEncode        0    actionSentDecode        0
  configureGood           2    configureFail           0
  encodeGood              0    decodeGood              2
  encodeBad               0    decodeBad               0
```

Table 31-14: Parameters in output of the **show enco counter=util** command

Parameter	Meaning
codeNullPacket	Number of utility jobs where the packet to be processed was null.
codeBadPacketLength	Number of utility jobs where the packet to be processed had a bad packet length.
actionSentEncode	Number of encode jobs started.
configureGood	Number of successful attempts to configure the utility channel.
encodeGood	Number of completed encode jobs.
encodeBad	Number of failed encode jobs.
codeBadPacketPriority	Number of utility jobs where the packet to be processed had a bad priority.
codeBadConfig	Number of utility jobs where the configuration was invalid.
actionSentDecode	Number of decode jobs started.
configureFail	Number of failed attempts to configure the utility channel.
decodeGood	Number of completed decode jobs.
decodeBad	Number of failed decode jobs.

show enco debug

Syntax `SHow ENCo DEBug`

Description This command executes a specific sequence of show commands to produce output useful for debugging. The specific commands are:

- **show enco**
- **show enco channel**
- **show enco counters** for all possible parameters

This command requires a user with Security Officer privilege when the switch is in security mode.

Examples To display information useful for debugging, use the command:

```
sh enc deb
```

Related Command
disable enco debugging
enable enco debugging
show enco
show enco channel
show enco counters

show enco key

Syntax `SHow ENCo KEY[=key-id]`

where *key-id* is a number from 0 to 65535

Description This command displays information about keys stored in the ENCO key memory. It requires a user with security officer privilege when the switch is in security mode.

If a key identification number is not specified, a summary of all keys stored in key memory is displayed (Figure 31-18 on page 31-46, Table 31-15 on page 31-46). If a key identification number is specified, only the specified key is displayed (Figure 31-19 on page 31-46).

Figure 31-18: Example output from the **show enco key** command

```
ID  Algorithm    Length Digest    Description              Module   IP Address
-------------------------------------------------------------------------------
 0  RSA-PRIVATE   768   E300FFDF  ROUTER KEY               -        -
 1  RSA-PRIVATE   512   DC5014A8  SSH Key                  SSH      -
 2  RSA-PUBLIC   1024   2A1596C9  CHCH router              -        192.168.2.1
 3  RSA-PUBLIC   1024   9348B823  Pauls key                -        -
 4  RSA-PUBLIC    512   C4A396A0  Carls Datafellow key     -        -
 5  DES            64   5A6798BD  Man key for CHCH SA      -        192.168.2.1
 6  3DES2KEY      128   45FE62AB  Man key for INV SA       -        192.168.3.1
 7  GENERAL       768   D56D323F  Auth key for INV         -        -
```

Table 31-15: Parameters in output of the **show enco key** command

Parameter	Meaning
ID	Identification number for the key.
Algorithm	Encryption algorithm for which the key was created: DES3DES2KEY 3DESINNER 3DESOUTER RSA-PRIVATE RSA-PUBLIC GENERAL
Length	Length of the key in bytes/bits.
Digest	MD5 digest of the key data.
Description	User-defined description for the key.
Module	Protocol associated with this key.
IP Address	IP address associated with this key.

Figure 31-19: Example output from the **show enco key** command for a specific DES key

```
ijn69p4v95e5qk
0x425BCFBF55FEC9B8
```

Examples

To display the list of all enco keys use the command:

```
sh enc key
```

To display a single DES key, use the command:

```
sh enc key=1
```

Related Commands

create enco key
destroy enco key
set enco key